Remarks/ Arguments

In the specification, paragraph [0002] has been amended for clarifying of field of invention. Paragraphs [0006], [0007], and [0008] have been amended for clearer contrast with known prior art. Paragraph [0020] has been amended to highlight the main utility provided by invention. Paragraph [0058] has been amended for clearer introduction to detailed description. Paragraphs [0017], [0018], [0019], [0021], [0022], [0024], [0025], [0059], [0079], [0081], [0098], [0107], [0111], [0114], [0116], [0122], and [0126] have been amended to reflect amendments made in claims 1-3 and 5-20 in regard to claims' subject matter.

Claim 4 has been canceled. Claims 1-3 and 5-20 remain in this application.

Claims 1 and 2 have been amended to provide clarify antecedent basis on occasions where Examiner finds it insufficient.

Claims 1-3 and 5-20 have been amended to be directed to patentable subject matters under 35 U.S.C. 101 and 35 U.S.C. 112. Claims 1-3 and 10 have been amended to claim "method" as subject matter. Claims 5-9 and 11-17 have been amended to claim "system" as subject matter. Claims 18-20 have been amended to claim "computer readable storage medium storing a set of instructions capable of being executed by a processor" as subject matter. Word "optionally" is not used in amended claim 2 and it is substituted with "where necessary" in amended claim 5. Limitations in claim 2 recited as "preferred embodiment", "this workflow-process", "might be executed", and "might initiate" are not used in amended claim 2.

In view of Examiner's rejection of claims 4 and 19 under 35 U.S.C. 102(b) as being anticipated by Cloud et al. (hereafter Cloud)(US Pat. 6,253,369), Applicant would like to draw attention to the following:

Cloud discloses art that incorporates user-interrogated information into a skeleton of source code and compiles it into a workflow object in form of executable code, which can be incorporated into a workflow where workflow object is the smallest unit of work capable of being executed. In this art, "a workflow contains one

synchronous unit of work and optionally one or more units of work that may be dispatched to execute substantially concurrently".

Despite cancellation of claim 4, the amended claim 9 is directed to "system for transactional processing of workflow as per claim 8, further including hierarchical structure of threads with four levels providing capacity for concurrent processing of multitude of workflow instances". Cloud's concurrent execution of multiple workflow objects denotes concurrent execution of objects within one workflow instance. However, concurrent execution of objects within one workflow instance is completely different art from the concurrent processing of multitude of workflow instances.

Applicant respectfully requests that subject matter of claim 9 is not rejected under 35 U.S.C. 102(b).

Claim 19 has been amended to drop the quoted characteristics "executes as operating system service", "uses a facility for message queuing and transactional messaging for signaling requests for execution of workflow-activities", and "uses a facility for distributed transaction coordination" as not being distinctive for the disclosed art. Amended claim 19 points out list of other distinctive characteristics, where the important one in regard to Cloud is the "capacity for concurrent processing of multitude of workflow instances as per claim 9", which is different from Cloud's "concurrent execution of objects".

Applicant respectfully requests that claim 19 is not rejected under 35 U.S.C. 102(b) and wishes to draw attention to the following:

According to Boston Consulting Group (BCG) publication [http://www.bcg.com/publications/files/B2Bresearch.pdf] in BCG Research Bulletin (December 1999), online negotiated US inter-company purchases will have value of $200 billion in year 2000 and, after experiencing annual growth rate of 58%, will reach the value of $1.3 trillion in year 2004. This forecast turned to be completely incorrect.

According to the same publication, 80% of interviewed US executives expected off-line negotiation to be almost always necessary for completion of inter-company online orders in year 2000. Despite the expressed need, even now, seven years later, no Web accessible facility for online negotiation of electronic purchases exists and no

technology capable of conducting online negotiation of electronic purchases is advertised as existing.

Presented retrospection to BCG forecast reveals a very unusual picture. Technology for online purchase negotiation is needed and anticipated. Market was ready to accept it even 7 years ago, but its creation has not been announced yet. How can this be explained? The answer is simple: Web-accessible business applications must have capacity to serve unpredictable high number of concurrent users. Main barrier in creation of Web applications, processing user interactions with execution of workflow, is the non-existence of application server platform (among the popular brands) capable of processing high-volume of workflow instances concurrently.

The most popular application servers capable of running workflow, like IBM's WebSphere and BEA's WebLogic, run in environment of Java Virtual Machine (JVM). However, as IBM admits, JVM application server process has inherent concurrency limitations [http://publib.boulder.ibm.com/infocenter/wasinfo/v4r0/index.jsp?topic=/com.ibm.websphere.v4.doc/wasa_content/07010303.html]. As a result, an instance of JVM application server is not capable of running concurrent workflow instances. Attempts to address concurrency limitation of Java technology were first announced in year 2005 with publication about research for development of Multi-Tasking Virtual Machine [http://java.sun.com/developer/technicalArticles/Programming/mvm/].

Microsoft recognized the importance of ability of Web applications to process user interactions with execution of workflow by announcing in August 2005 commencement of development of Windows Workflow Foundation (WWF). However, until now there's no official statement from Microsoft that WWF is, or will be, capable to execute concurrent workflow instances.

Moreover, as Herb Sutter of Microsoft noticed in year 2005 about entire software industry in context of anticipated concurrency revolution: "The state of concurrency in the software industry is terrible. ... The worlds best frameworks are just getting away with it." [www.devconnections.com/updates/LasVegasFall_05/CPP_Connections/CPPKEY2_Software%20and%20Concurrency.pdf]

In contrast, in year 2003, the disclosed in this application art regarding workflow concurrency was used to build the only available at present on the market technology for processing of high-volume, 1000 per computer processor, concurrent workflow instances.

In view of Examiner's rejection of claims 1 and 18 under 35 U.S.C. 102(e) as being anticipated by the applicant admitted prior art (hereafter AAPA), Applicant would like to draw attention to the following:

Mentioned in paragraph [0007] documents demonstrate prior art related to concept of workflow engines interpreting workflow graph description for execution of software components, representing workflow activities, in described sequence. Paragraph [0011] comments the strong and weak aspects of concept of template-based source code development. It points out why this concept, despite its usefulness, cannot substitute the need for pure generation of source code. Paragraph [0010] comments the strong and weak aspects of prior art related to declarative programming for creation of data-driven executable code and why, despite the degree of flexibility it adds to executable code, this art is not applicable for automated development of workflow application with practically unlimited number of possible workflow configurations.

Claim 1 discloses art, which provides means for graphical development of executable code for any possible workflow configuration. This art enables development of executable workflow application with no human code writing. Art for development of executable workflow application without human code writing is not disclosed in AAPA. Furthermore, the discussed in paragraph [0007] execution of workflow components, where said components might or might not be in form of executable code, does not make a workflow application in form of workflow engine, script, and components an executable workflow application as it is in claim 18.

Applicant respectfully requests that claims 1 and 18 are not rejected under 35 U.S.C. 102(e).

In view of Examiner's rejection of claims 2 and 3 under 35 U.S.C. 103(a) as being unpatentable over AAPA in view of Kim et al. (hereafter Kim)(US Pub. 2002/0065701), Applicant would like to draw attention to the following:

Kim discloses art, which is part of AAPA.

In Kim disclosed art, business process is defined in terms of its "administrative steps and properties". In the art's preferred embodiment, the "property data" is "in an XML format using Workflow Definition Language (WfDL) of Workflow Management Coalition (WfMC)". Kim does not disclose a method for defining a workflow process.

Even though it is not explicitly mentioned in Kim or the rest of AAPA, it might be assumed that specification of business process property data in the art presented in Kim and the rest of AAPA contains a flow graph of a workflow process. Inherent steps of defining a flow graph of a workflow process are: firstly to specify the workflow-activity that must be executed first, then to specify branches of flow of control based on possible results of its execution, then the next workflow-activity that might be executed depending on result of execution of previous workflow-activity, and so on until the last workflow-activity is specified. This is exactly the way workflow definition is presented in XML format - as a step-by-step representation of nodes and arrows of workflow graph with XML language.

In contrast, the method for defining of workflow process as presented in claim 2 does not follow the above-mentioned inherent steps; neither the produced definition is in XML format using Workflow Definition Language (WfDL). Even though a created with this method definition might be regarded as just an alternative way of conveying data describing a workflow graph, the produced workflow graph definition in form of matrixes is the only convenient form of presenting workflow graph data to the program that uses this data as input for producing source code of an executable workflow application as output.

Applicant respectfully requests that claim 2 is not rejected under 35 U.S.C. 103(a).

Art presented in claim 3 is related to generation of source code for executable workflow application. Concept and practices behind AAPA disclosed declarative programming discussed in paragraph [0009] and concept and practices behind generation of source code are not identical. Declarative programming might involve customization of code at compile time or customization of code at runtime. Template-based development is declarative programming with code customization at compile time. Data-driven code exemplifies declarative programming where code

customization happens at runtime. As argued in paragraph [0011], unlike with data-driven programming, not all software variations could be expressed with templates. Data-driven programming involves writing of source code that is created once, compiled, and built to produce data-driven executable code. During its execution, data parameters are passed to the executable code for interpretation, thereby controlling the behavior of executable code.

In contrast, the process of generation of source code comprises interpretation of input data to produce unique source code, which will later be used to compile and link into unique executable code. The customization of code happens at compile time, like with template-based programming and thereby it is more efficient then the data-driven code. In addition, generation of source code can express all necessary variations, which cannot be achieved with templates. Thereby, generation of source code delivers the combined advantages of both types of declarative programming without having anyone of their shortcomings. Art presented in claim 3 is dominantly based on source code generation.

Applicant respectfully requests that claim 3 is not rejected under 35 U.S.C. 103(a).

In view of Examiner's rejection of claim 5 under 35 U.S.C. 103(a) as being unpatentable over Cloud as applied to claim 4, and further in view of Du et al. (hereafter Du)(US Pat. 6,308,163), Applicant would like to draw attention to the following:

Related to claim 5 art discloses hierarchical tree of class objects capable of representing each individual workflow configuration from the set of virtually unlimited number of possible workflow configurations. Every time it is constructed, the hierarchical tree of class objects represents one particular workflow configuration within the borders of the workflow process that constructed it and has no meaning outside the borders of this particular workflow process.

Art disclosed in Du is a "method and a system for providing resource management in workflow processing of an enterprise include a multi-level resource manager hierarchy", where a resource manager "manages workflow resources (i.e., keeps track of status of resources) and assigns workflow resources to business steps (i.e., process activities) when requested to do so by a workflow execution engine".

The multi-level resource manager hierarchy coordinates "wide distribution of enterprise workflow resources across organizational and physical boundaries" via global resource managers, enterprise resource managers, and site resource managers communicating between themselves with exchange of messages.

In the art disclosed in Du, a resource manager is individual process running on individual physical machine. Even though each resource manager being member of multi-level hierarchy might have own class objects, the entire set of class objects belonging to resource managers cannot be regarded as hierarchy of class objects. The notion of class objects is meaningful only within the process borders and meaningless outside. Therefore, two or more processes cannot construct a common hierarchy of class objects; neither a hierarchy of processes with class objects might be regarded as hierarchy of class objects.

The hierarchical tree of class objects of claim 5 does not provide the benefit of having an efficient management system facilitating the flow of workflow resources as does the art disclosed in Du. In any particular workflow process in form of executable workflow application, the hierarchical tree of class objects simply represents the configuration of its workflow. As part of this invention, however, the hierarchical tree of class objects provides the benefit of ability to represent a virtually unlimited set of possible workflow configurations. Cloud discloses executable workflow objects and does not disclose executable workflow application; any hierarchical tree of class objects representing a particular workflow application is meaningless outside the borders of executable process that is executable workflow application. Art of Cloud and art of Du combined together do not hint or suggest a hierarchy, or any other configuration, of class objects capable of representing each individual workflow configuration from the set of virtually unlimited number of possible workflow configurations.

Applicant respectfully requests that claim 5 is not rejected under 35 U.S.C. 103(a).

In view of Examiner's rejection of claims 6-9 under 35 U.S.C. 103(a) as being unpatentable over Cloud and Du as applied to claim 5 above, and further in view of Hsu et al. (hereafter Hsu)(US Pat. 5,581,691), Applicant would like to draw attention to the following:

Related to claim 6 art discloses splitting of a control flow by launching one or more new parallel control flows.

In the art disclosed in Hsu, "work flow can include parallel work flow paths". It says nothing whether a parallel control flow can launch new parallel control flow. It does not discuss possible implications related to launching of a parallel control flow from a parallel control flow. Nor does it provide a solution to these implications.

In contrast, the presented in application figure 3 shows a workflow-activity from a parallel control flow A23 launching new parallel control flow A32. As it is discussed in paragraphs [0085] and [0086] and as it is presented on figure 11, launching of a parallel control flow and especially launching of a parallel control flow from a parallel control flow necessitates inclusion of notification-connectors and logical elements 1136 and 1155. Notification connectors are presented with tick lines on figure 11. They signal unsuccessful execution of a workflow-activity that participates in a parallel flow of control. Where necessary, notification messages trigger execution of compensators - workflow-activities executed to reverse already committed ACID transactions. Moreover, notification messages trigger unconditional purging of messages belonging to the workflow instance that is being terminated from message queues used for synchronization. Without this purging mechanism, synchronization message queues will be polluted with messages belonging to terminated workflow instances and the speed of execution of application's concurrent workflow instances will gradually degrade in time.

The presented in above paragraph is a substantially new art that differentiate the subject matter of claim 6 from the disclosed in Hsu. In regard to Cloud, the disclosed concurrent execution of multiple workflow objects does not launch new parallel control flows within the borders of a process, which is executable workflow application. In regard to Du, the disclosed multi-level resource manager hierarchy in no way might be presented in context of parallel control flows or in context of concurrent execution of multiple workflow objects disclosed in Cloud. Arts of Cloud, Du, and Hsu are each one individually, or taken together, related to integration of heterogeneous computer systems. Nothing in their combined art gives any hint about problems starting, and restrictions applicable, with launching of a parallel control flow and especially with launching of a parallel control flow from a parallel control, and nothing in their combined art gives a hint on possible technical solution for problems associated with launching of a parallel control flow.

Applicant respectfully requests that claim 6 is not rejected under 35 U.S.C. 103(a).

Related to claim 7 art discloses synchronization of two or more parallel control flows before execution of next in flow-graph workflow-activity according to a synchronization scheme.

Hsu does not disclose any synchronization of parallel paths (control flows). Instead, it just says that "work flow can include parallel work flow paths". Arts of Cloud, Du and Hsu individually or combined together do not disclose or suggest anything related to synchronization of parallel path or control flows. In respect to Hsu, it appears logical to assume that once parallel pats exist they must be synchronized somewhere prior to execution of next workflow-activity. However, nothing in art of Hsu, explicitly or implicitly, suggests that there must be execution of at least one workflow-activity after execution of parallel paths completes.

In contrast, application figure 11 shows in details claimed synchronization apparatus. Logical element 1136 complements functionality of logical element 1135 and logical element 1155 complements functionality of logical element 1154. Figure 11 is a detailed view of workflow presented on figure 3. During execution of an instance of workflow shown on figure 3, if execution of A23 fails, workflow instance will terminate and message queues of synchronization threads that trigger execution of A31 will contain message that eventually will stay there until application is shut down and message manually removed. If execution of A21 or A22 fails, workflow instance will terminate and belonging to it messages will be presented in message queues of synchronization threads that trigger execution of A31 and A51. Presence of messages there will degrade application performance.

To cope with conditions like this, synchronization apparatus of claim7 is more then just logical 'AND' of message that relate to a particular workflow instance from the multitude of all concurrently executed instances and on result TRUE - triggering of execution of next workflow-activity. As it is shown on figure 11, failure of execution of A22 will send notification to queue 1134b instead of message in queue 1133b. This notification will trigger purging of message received in queue 1133a as a result of successful execution of A21 and message received in queue 1133c as a result of successful execution of A23 and will trigger generation and sending of a

notification to queue 1153a. Successful execution of A23 will trigger execution of A32, which after being successfully executed will send a message to queue 1152b. This message would have stayed there until application is shut down and message is manually removed. To prevent this from happening, a mechanism triggered by notification received in 1134b will generate and send a notification to 1153a. On receiving notification in 1153a, logical element 1155 triggers purging of message received in 1152b from A32.

Even if arts of Cloud, Du and Hsu, individually or combined together, were disclosing or suggesting any simple synchronization mechanism, the synchronization apparatus that is subject matter of claim 7 would have been different as it deals with wider range of problems sourcing from synchronization. Cloud, Du and Hsu, however, individually or together, do not suggest anything related to synchronization of parallel path or control flows.

Applicant respectfully requests that claim 7 is not rejected under 35 U.S.C. 103(a).

Related to claim 8 art discloses launching of alternative control flow routes. An alternative control flow route might be confused with concurrent control flow route since both appear parallel on drawings. However, while concurrent control flows are initiated by split of control flow sourcing from the same workflow-activity output, an alternative control flow route starts from workflow-activity output that is different from the output that is source of the normal control flow route. Additional difference is the point of merging of the alternative flow of control with the normal control flow route. The trigger on point of merging is logical 'OR' and there is no need for synchronization between both flows of control as either the normal either the alternative flow of control route will be used by a workflow instance at any point of time. The last difference between alternative and concurrent control flow routes is the fact that an alternative control flow route does not have to execute a workflow-activity; it might exists only for skipping execution of one or more workflow-activities located on the normal execution route. In contrast, an existence of concurrent control flow route with no workflow-activity executed is completely meaningless.

On figure 12 of application, an alternative control flow is the one with number 1245. This purpose of this figure is not only to show an alternative control flow. This

figure illustrates the more complex mechanics of synchronization of concurrent control flows and related to it logical scheme 1236 for notifications and purging of messages from synchronization message queues. Paragraphs [0088] and [0097] explains the required synchronization mechanism where an alternative control flow route bypasses execution of a workflow-activity that is triggered immediately after synchronization of few concurrent control flows. Arts of Cloud, Du and Hsu, individually or combined together, do not explicitly disclose or implicitly hint any art similar to application art that relates to alternative control flows.

Applicant respectfully requests that claim 8 is not rejected under 35 U.S.C. 103(a).

Related to claim 9 art discloses hierarchical structure of threads with four levels providing capacity for concurrent processing of multitude of workflow instances.

Hsu discloses art "concurrently executing a plurality of processes". In the art of Hsu, execution of workflow comprises execution of steps, wherein each step is an individual application program. Execution of workflow steps is coordinated by flow controller. The controller is an individual program (or process). This art relies on ability of computer operating system to run multiple application programs (processes) or multiple instances of an application program (process). The controller starts execution of a workflow step by starting execution of the relevant application program that represents this step. Thus, the controller can start concurrent execution of more then one workflows. The number of concurrently executed workflows is limited by number of application programs that operating system can execute concurrently, i.e. "the number of processors to which the flow controller can assign work". Workflow concurrency that relies on operating systems running multiple processes has very restricted, and from today's prospective very low, upper limits.

In contrast, in regard to claim 9, application art provides capacity for workflow concurrency that can be scaled up to 1,000 concurrent workflow instances per processor (CPU). Its principal difference with the art presented in Hsu is in the fact that processing these 1,000 concurrent workflow instances per CPU is enacted by a hierarchy of threads that belongs to one workflow process, which is an instance of a workflow application. For example, on computer with 2 CPUs a single workflow

process can concurrently process 2,000 workflow requests of same type. Or, for example, on the same computer with 2 CPUs, one workflow process can concurrently process 1,000 workflow requests of same type, a second workflow process can concurrently process 500 workflow requests of same type, and the other 5 workflow processes can concurrently process 100 workflow requests of same type each.

Technical field of art in Cloud is integration of heterogeneous computer systems. Cloud's concurrent execution of multiple workflow objects denotes concurrent execution of objects within one workflow instance, whereas subject matter of claim 6 is hierarchical structure of threads with four levels providing capacity for concurrent processing of multitude of workflow instances. Moreover, the concurrent execution of workflow objects in art of Cloud do not happen in one computer system; therefore, it is not performed by one process.

Technical field of art in Du is resource management in workflow processing of an enterprise. Du discloses a multi-level resource manager hierarchy, where a resource manager "manages workflow resources (i.e., keeps track of status of resources) and assigns workflow resources to business steps. In this art, a resource manager is individual process running on individual physical machine. Even though each resource manager being member of multi-level hierarchy might have own thread, the entire set of thread belonging to resource managers cannot be regarded as hierarchy of threads.

In arts of Cloud, Du, and Hsu, workflow is performed by multiple processes. The notion of thread is meaningful only within the process borders and meaningless outside. Therefore, two or more processes cannot construct a common hierarchy of thread; neither a hierarchy of processes with threads might be regarded as hierarchy of threads. Furthermore, Cloud, Du, and Hsu do not explicitly mention term "thread" and individually or together do not disclose any art that might be related to, or used as hunt for construction of, hierarchical structure of threads.

Applicant respectfully requests that claim 9 is not rejected under 35 U.S.C. 103(a).

In view of Examiner's rejection of claims 10-16 under 35 U.S.C. 103(a) as being unpatentable over Cloud, Du and Hsu as applied to claim 9 above, and further

in view of LiVecchi (hereafter LiVecchi)(US Pat. 6,823,515), Applicant would like to draw attention to the following:

Related to claim 10 art discloses a method for transactional plugging of software component into workflow-activity of workflow-process at run-time. Plugging of software components in the body of executable application is generally described as instantiation of component class in the process space of executable application for invoking of component methods by threads belonging to process space of executable application.

The concept behind claim 10 of application is described in paragraph [0107]. In the preferred embodiment of our application, the method of claim 10 uses services of COM+. Per each workflow-activity, it involves usage of pair of a generic non-transactional and a generic transactional component installed on a physical computer and registered with a COM+ application. A COM+ application is an individual process with its own boundaries. All transactional components providing business functionality of the workflow-activities must be installed and registered with the same COM+ application as well.

When a workflow application attempts to instantiate a transactional component registered with a COM+ application, COM+ does not instantiate it. What is instantiated is just an object that intercepts requests for instantiation of transactional component. In fact a transactional component object is instantiated within the boundaries of a COM+ application only when a transaction is requested and it is destroyed when the transaction is committed of aborted. Due to the very limited life span of transactional component objects, they cannot be used to store any data related to configuration of the workflow application - such as sourcing MSMQ queue (or queues when synchronization is involved) and the entire set of alternative destination MSMQ queues that will be used depending on result of execution of workflow-activity for serving a particular workflow request. Consequently, data that is not specific or distinctive to any particular transaction must be transmitted over process boundaries of the COM+ application each time when a workflow-activity is executed. Moreover, handles to all sourcing and destination MSMQ queues must be created inside the boundaries of COM+ application every time a workflow-activity is executed.

To deal with this inefficiency, the art of claim 10 introduces generic non-transactional components - registered with and instantiated in the COM+ application. A generic non-transactional component is a special kind of initialization and data holding component, paired with a generic transactional component. A generic transactional component is link of the chain of a particular workflow instance. Execution of a workflow instance moves one-step forward after successful completion of transaction initiated by a generic transactional component. A generic transactional component is a socked where transactional components that provide functionality of workflow-activity are being 'plugged'. The 'plugged' workflow-activity-related transactional components will be executed as part of the initiated transaction and will participate in voting for its completion or abortion.

Invention of LiVecchi is related to enhancing performance of a computer running a multithreaded server application. Its does not, however, explicitly mention or implicitly deal with concepts related to terms 'workflow', 'workflow-activity', 'activity', 'transactional component', and 'non-transactional component'. LiVecchi's art is not related to transactional workflow or transactional workflow applications in any aspect.

Arts of Cloud, Du and Hsu are related to workflow. However, arts of Cloud, Du and Hsu are not related in any aspect to executable workflow applications and particularly to multithreading features of executable workflow applications. Knowledge derived from Cloud, Du and Hsu in regard to enterprise workflow can in no way be implemented in conjunction with knowledge derived from LiVecchi in regard to multithreaded server application dealing with work balancing related to receiving and sending messages over passive socket.

Common characteristic of arts of Cloud, Du, Hsu and LiVecchi is that neither of these arts, individually or taken together, deals with the concept of workflow components in general, and with transactional plugging of workflow components and associated to it inefficiencies, in particular. Therefore, individually or combined do not produce knowledge related transactional plugging of workflow components.

Applicant respectfully requests that claim 10 is not rejected under 35 U.S.C. 103(a).

Related to claim 11 art discloses workload balancing structured at two levels. Upper level comprises multitude of associations between a dispatching thread and multitude of supervising threads and involves dispatching thread balancing workload between its associated supervising threads. Lower level of comprises multiple groupings of processing threads in pools associated with a supervising thread per pool and involves supervising threads balancing workload between processing threads of their associated pools.

Presented in LiVecchi art discloses a "scheduling heuristic" that "is defined for optimizing the number of available threads". When it comes to present the criteria of said optimization, LiVecchi discloses that goal of the "scheduling heuristic" is "to alleviate over-scheduling of said worker threads", i.e. to minimize the number of unnecessary switching between active worker threads, whereas "number of threads to be created for the pool is typically specified by a user (e.g. a systems administrator), as a configuration parameter when initializing the server". In context of real-time operating systems, scheduling denotes the procedure of making a thread active and using a time slice of processor time. The problem with over-scheduling relates to inefficiency in using system resources; for example, processor time for switching on and off some active threads that do not perform anything useful.

In its essence, the concept behind LiVecchi's scheduling heuristic is the following: When a new connection is established, instead of activating a thread from the pool of blocked threads to start processing the incoming call, it is better to wait for a while expecting that one of the active thread will complete its current job and can start processing the waiting incoming call. Thereby, the number of unnecessary active threads will stay minimal and this minimal number of excessive active threads is paid with the small delay before incoming calls are started being processed.

LiVecchi states that "scheduling heuristic further comprises a subprocess for balancing". Therefore, mentioned balancing should not be regarded outside the concept of scheduling heuristic. LiVecchi provides another hint in the same direction by specifying "balancing may further comprise using an average delay, and also a maximum delay".

To summarize, in LiVecchi, the total number of worker threads in the pool is specified by the system administrator before server process is started. Workload balancing relates to minimization of number of unnecessary active threads. It does

not relate to total number of worker threads, as it is constant during process execution.

In contrast, in what relates to claim 11 art, total number of worker threads in the plurality of pools is dynamically defined at runtime by the workflow application. Number of active worker threads is never higher then the number of processed requests since a worker thread is blocked immediately after it completes with execution of a workflow-activity.

LiVecchi acknowledges the problem that "On SMP (Symmetric Multiprocessor) machines, the dispatcher thread may become a bottleneck that prevents worker threads from being scheduled fast enough to keep all of the processors busy". In this sentence, term 'scheduled' means activated, loaded with work, and participating in the scheduling. He acknowledges that attempts to avoid this problem with the "new interrupt approach leads to the first performance problem to be addressed by the present invention, which will be referred to herein as 'over-scheduling'".

In contrast, art that relates to claim 11 of application introduces a new approach to the problem with bottlenecks related to dispatcher threads. This new approach seeks the solution in exactly the opposite direction. Instead of eliminating the function of dispatcher threads with "use of event-driven interrupts" leading to "over-scheduling", it introduces a new layer of threads between a dispatcher thread and its pool of worker threads. It divides the pool of worker threads belonging to the dispatcher into smaller pools of worker threads, wherein each pools of worker threads is headed by a supervising thread. This layer of supervising threads takes the burden of supervising work from dispatcher. As a result, the only responsibility of the dispatcher thread is to dispatch incoming work equally between supervising threads and it never gets overloaded with work and never becomes a bottleneck.

Arts of Cloud, Du and Hsu are related to workflow. However, arts of Cloud, Du and Hsu are not related in any aspect to executable workflow applications, to bottlenecks in executable workflow applications, and to workload balancing in executable workflow applications. Knowledge derived from Cloud, Du and Hsu relates to management of internal enterprise workflow. It cannot be used in conjunction with knowledge derived from LiVecchi in regard to balancing of work related to receiving and sending messages over passive socket for the reason that knowledge of

enterprise workflow management does not have a common intersection with knowledge of multithreaded socket level programming.

Applicant respectfully requests that claim 11 is not rejected under 35 U.S.C. 103(a).

Related to claim 12 art discloses prevention and neutralizing of software bottlenecks involving encapsulation of a thread pool containing fixed number of processing threads with a supervising thread in a processing-pipe, construction of additional processing-pipes, and inclusion of constructed additional processing-pipes in workload balancing process related to workflow-activity where development of bottleneck has been detected. Whenever a workflow-activity experiences processing delays, it becomes a bottleneck in the workflow application simultaneously processing multitude of workflow instances. To cope with the insufficient processing power dedicated to this activity, application art dynamically creates and assigns processing resources to this workflow-activity.

LiVechi discloses creation of "collector socket" where "Input from multiple passive sockets is merged onto a collector socket, so that a single source is available from which to schedule connections to threads". Collection socked is created to facilitate unified processing of messages received from multiple ports over the same IP address [example: online shopping application receiving messages over HTTP and HTTPS], or from the same port over multiple IP addresses [when "a Web server is hosting more than one hostname, each hostname having its own IP address"]. It does facilitate input/output operations. Facilitation of input/output operations, however, is not related to processing of user-input. Moreover, in LiVechi, processing of incoming user-input, wherever it is mentioned, does not involve execution of workflow and work cannot be related to bottlenecks prevention in workflow systems. Therefore, creation of "collector socket" is in now way related to prevention and neutralizing of software bottlenecks in a workflow system.

Combined knowledge of Cloud, Du and Hsu is related to workflow management but it is not related to quantitative aspects of processing of high-volume concurrent workflow, and, therefore, is not related to bottlenecks prevention in workflow systems. Furthermore, it cannot be used in conjunction with knowledge derived from LiVecchi in regard to creation of "collector socket" where "Input from multiple passive sockets is merged onto a collector socket, so that a single source is

available from which to schedule connections to threads", for the reason that the problem solved by LiVecchi is related to unified handling of input/output operations and does not intersect with workflow management. Therefore, knowledge of Cloud, Du and Hsu cannot be combined with knowledge derived from LiVecchi to produce art for prevention and neutralizing of software bottlenecks in a workflow system.

Applicant respectfully requests that claim 12 is not rejected under 35 U.S.C. 103(a).

Related to claim 13 art discloses detection of conditions requiring scaling up of available capacity for processing of concurrent workflow.

LiVecchi discloses a scheduling heuristic used to determine whether more threads should be activated to process incoming messages. In LiVecchi art, the total number of worker threads in the pool is specified by the system administrator. The system administrator specifies the total number of worker threads in the pool before server process is started. The total number of worker threads in the pool stays constant during process execution. The scheduling heuristic aims minimization of number of unnecessary active threads. The scheduling heuristic saves the processor time that otherwise is wasted for switching of activated but idle threads. The scheduling heuristic, however, cannot save the system resources wasted for creation of the threads that will never be used.

In contrast, in what relates to claim 13, application art discloses detection of critically small number of blocked threads dedicated to a particular workflow-activity. This detection aims to maintain a guaranteed minimal reserve of blocked threads dedicated to this workflow-activity, as the total number of worker threads is not specified before process execution but established dynamically. Unlike LiVecchi's constant number of total worker threads, this art is used to increase applications' number of total worker threads and let two or more applications compete for the free system resources only when these resources are needed.

Combined knowledge of Cloud, Du and Hsu is not related to quantitative aspects of processing of high-volume concurrent workflow, and is not related to scaling of available capacity for processing of concurrent workflow. Therefore, it does not hint, individually of in conjunction with LiVecchi, anything related to detection of

conditions requiring scaling up of available capacity for processing of concurrent workflow.

Applicant respectfully requests that claim 13 is not rejected under 35 U.S.C. 103(a).

Related to claim 14 art discloses apparatus for scaling up of available capacity for processing of concurrent workflow.

LiVecchi discloses creation of "collector socket, so that a single source is available from which to schedule connections to threads". With other words, collector socket is created "for merging input from more than one source and making that merged input available for scheduling". Scheduling of a connection denotes assigning of this connection to a particular thread for receiving and processing of the incoming message over this connection. The collector socked is created to deal with the "multiple input source' problem. It is not related to alleviating the problem of over scheduling of threads.

According to LiVecchi: "On SMP (Symmetric Multiprocessor) machines, the dispatcher thread may become a bottleneck that prevents worker threads from being scheduled fast enough to keep all of the processors busy". When in order to avoid this problem "dispatcher threads are not used, and the responsibility for checking the arrival queue belongs with the worker threads, the thread pool will be statically partitioned across the set of passive socket queues". "Because the workload at any particular time, and the corresponding distribution of requests among the passive sockets, is unpredictable, it is very likely that this static partitioning will be less than optimal." "When too few threads are available, incoming requests have to wait on the queue, while available system capacity is left idle".

The partitioning of thread pool and the "use of event-driven interrupts" for processing of incoming connections creates the problem of over-scheduling of threads on some HTTP queues. Over-scheduling does not cause delays and software bottlenecks. As a matter of fact, it is exactly the opposite. Substitution of dispatcher threads with the event-driven interrupts when applied on partitioned thread pool might cause bottleneck conditions. For example, when number of threads in a particular partition is insufficient to handle a surge of communication activities over the socked associated with this partition. Therefore, bottlenecks result from under-

scheduling and a solution that prevents over-scheduling does not prevent bottlenecks.

In what relates to claim 14, application art discloses usage of dispatcher thread and a new layer of supervising threads that prevents dispatcher thread from becoming a bottleneck. It discloses workflow application scaling-up at a particular workflow-activity to counteract processing delays at this workflow-activity and prevent it from becoming a bottleneck in the workflow application simultaneously processing multitude of workflow instances. Furthermore, it discloses automatic scaling-up with all of application workflow-activities, triggered to cope with increased workload. Automatic scaling-up at a particular workflow-activity involves creation of an additional processing-pipe and inclusion of created processing-pipe in workflow balancing. A processing-pipe is a group of a supervising thread and fixed number of worker threads. Thereby, the total number of worker threads of a workflow application is dynamically increased when triggering conditions are detected.

LiVecchi does not mention or specify a mechanism for prevention of bottlenecks development and does not disclose scaling-up at runtime. Under LiVecchi art, the system administrator must specify the total number of worker threads in the pool before the process is started and the total number of worker threads in the pool cannot be dynamically modified. In addition, combined knowledge of Cloud, Du and Hsu is not related to quantitative aspects of workflow processing and cannot individually, or in conjunction with LiVecchi, hint anything having similarity with scaling up of available capacity for processing of concurrent workflow.

Applicant respectfully requests that claim 14 is not rejected under 35 U.S.C. 103(a).

Related to claim 15 art discloses detection of conditions requiring scaling down of available capacity for processing of concurrent workflow.

LiVecchi discloses a detection of condition when an idle persistent connection over a socked should be removed and the busy thread associated to this connection will be released back to application pool. This detection is based on "exceeding a maximum number of idle connections". LiVecchi does not explicitly or implicitly

detects conditions when total number of threads in application pool needs to be decreased, as this number is constant during application running.

In contrast, claim 15 art discloses detection of condition requiring scaling-down the total number of threads in application pool. This detection is based on conjunction of events from all processing pipes associated to a workflow-activity. Each one of these events signals that number of busy threads in a processing-pipe reached its critical minimum. As a result of this detection, the total number of threads in application pool will be decreased with the number of worker threads belonging to one processing-pipe.

Combined knowledge of Cloud, Du and Hsu is not related to quantitative aspects of processing of high-volume concurrent workflow, and is not related to scaling of available capacity for processing of concurrent workflow. Therefore, it does not hint, individually of in conjunction with LiVecchi, anything related to detection of conditions requiring scaling down of available capacity for processing of concurrent workflow.

Applicant respectfully requests that claim 15 is not rejected under 35 U.S.C. 103(a).

Related to claim 16 art discloses detection of conditions requiring scaling down of available capacity for processing of concurrent workflow.

LiVecchi discloses closing down of an idle persistent connection. Further, the busy thread associated to this connection will be released. As a result, the number of busy threads would be at a minimum. However, the total number of worker threads in application pool is specified by the system administrator before the application is started. As a result, even though the number of busy threads will be at a minimum, the unused threads of application pool will continue consuming system resources that cannot be made available to other applications running on the same computer.

In contrast, claim 16 art discloses apparatus for scaling-down of workflow application. This scaling-down dynamically decrements the number of processing-pipes in the application and, thereby, decreases the total number of threads in application pool. As a result, the system resources, consumed by the threads of the

released processing-pipe, are returned back to the system and are made available to other applications running on the same computer.

Combined knowledge of Cloud, Du and Hsu is not related to quantitative aspects of workflow processing and cannot individually, or in conjunction with LiVecchi, hint a solution that might resemble apparatus for scaling down of available capacity for processing of concurrent workflow.

Applicant respectfully requests that claim 16 is not rejected under 35 U.S.C. 103(a).

In view of Examiner's rejection of claims 17 and 20 under 35 U.S.C. 103(a) as being unpatentable over Cloud, Du, Hsu, and LiVecchi as applied to claim 16 above, and further in view of Lewis et al. (hereafter Lewis)(US Pat. 6,434,714), Applicant would like to draw attention to the following:

Related to claim 17 art discloses real-time visualization of quantity, structure, and utilization of hierarchical structure of threads and its adaptation-enacted modifications, where the real-time visualization serves as indicator of workload volume, indicator of points of delay caused by distributed infrastructure, and for observation and analysis of adaptive behavior of hierarchical structure of threads of already deployed and running workflow application.

Lewis discloses art that collects "performance data from hardware and software components of an application program, allowing a developer to understand how performance data relates to each thread of a program" and displays "an image comprising a graph for each thread reflecting the measuring period, a performance level corresponding to the predetermined performance-related aspect of the application program, and changes in performance at points in time in each state". To collect the necessary data, commands are, manually or automatically, inserted on appropriate places in thread's code. Collected data is visualized off-line by performance analyzer, which might show timing of thread's entering and leaving its states, performance criteria corresponding to each state, and portions of software code responsible for visualized performance.

In distinction, art that relates to claims 17 and 20 does not make any individual thread emit data related: to its internal states; to timing of getting into

thread states; or to performance criteria of each state. This art is in no way related to what is going on inside a thread during its execution. What is being collected by poling a workflow application is readily available generalized data: (a) about hierarchical structure of threads as reflection of workflow graph of a workflow application, (b) about quantity of threads dedicated to each one of workflow-activities, and (c) about degree of utilization of threads dedicated to each workflow-activity. The collected data is used for real-time visualization of application actions undertaken by apparatus for bottlenecks prevention and application scaling-up, and by apparatus for application scaling-down aiming to improve utilization of application threads.

Unlike the disclosed in Lewis art for collecting performance data of individual threads and for displaying an "image comprising a graph for each thread reflecting the measuring period", the art that relates to claims 17 and 20 collects and visualizes in real-time generalized data regarding quantity, utilization, and distribution of groups of hierarchical structures of threads among application's workflow-activities at points in time, where the importance of collected and visualized data is related to workflow application's ability to process high-volume of concurrent workflow instances.

Furthermore, the disclosed in Lewis art is applicable for fine tuning the performance of thread's code during the debugging stage of software development. While art related to claims 17 and 20 is applicable for measuring at runtime of ability already deployed automatically created workflow application to cope with spikes in volume of incoming concurrent workflow requests, to automatically neutralize development of bottlenecks inside the workflow, and to utilize consumed system resources.

Arts disclosed in Cloud, Du, and Hsu do not provide any knowledge related to performance of threads, quantity of threads involved in processing of workload, or visualization. Art of LiVecchi discloses supervision and optimization of number of activated threads. However, no real-time visualization or interpretation of said number of activated threads outside the context of scheduling heuristic is disclosed. Knowledge derived from arts of Cloud, Du, and Hsu, knowledge derived from art of LiVecchi, and knowledge derived from art of Lewis, combined or individually, do not explicitly suggest or implicitly hint a solution having a degree of similarity with real-time visualization of quantity, structure, and utilization of hierarchical structure of

threads and its adaptation-enacted modifications, where the real-time visualization serves as indicator of volume of concurrently processed workflow instances, indicator of points of delay caused by distributed infrastructure, and for observation and analysis of adaptive behavior of hierarchical structure of threads..

Applicant respectfully requests that claims 17 and 20 are not rejected under 35 U.S.C. 103(a).

Applicant respectfully requests allowing of amended claims 1-3 and amended claims 5-20.

Respectfully submitted,

By ....

Ivan I. Klianev
Applicant
Tel.: (61-2) 9567-1150
Fax: (61-2) 9567-1160